VI & VIC

Nr. 1 - 2. årg. - Februar-marts 1983 — Pris 10,95 kr.

Commodore 64 har større skærm, forbedret grafik og lyd - og en hukommelse på 64 K

I det ydre er der næsten ingen forskel på VIC-20 og Commodore 64, som nu præsenteres af Commodore Data også i Danmark.

Men denne nye computer har på en række områder egenskaber, som VIC ikke kan hamle op med: Ikke mindst inden for de grafiske og musikalske områder brillerer Commodore 64, og skærmen har plads til 40 tegn på hver linie. Sammen med det forbedrede skærmbillede er det især den større hukommelse, der gør 64'eren til et meget spændende tilbud for den seriøse computerbruger, men samtidig har den virkelig meget at byde på for den, der vil lege med farver, lyd og en helt ny grafik-mulighed, de såkaldte SPRITES.

Samme BASIC

For at tage lighederne først: Commodore 64 er forsynet med den samme BASIC-version, som VIC-20, så VIC-brugere vil uden videre kunne gå i gang med at programmere Commodore 64. Der vil blive mulighed for at forsyne maskinen med et COMAL-indstiksmodul for programmører, der foretrækker dette sprog.

Tastaturet på Commodore 64 er det samme, som kendes fra VIC-20, bortset fra, at funktionstasterne har fået en anden farve.

SPRITE

Ikke mindst spil-entusiaster vil glæde sig over de grafiske muligheder, Commodore 64



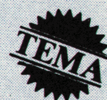
Sådan kan ét program bruge et andet

Læs side 2 og 3

VIC-20 hjertet i HI-FI-demo system

Læs side 6

7 sider om maskinsprog på VIC-20



Læs side 7-13

Fortsat fra forsiden

byder på. En SPRITE er en brugerdefineret figur, som i helt glidende bevægelser kan styres rundt på skærmen - ja, man kan endda styre dem lidt uden for det synlige skærmbillede. Der kan på samme tid defineres otte sådanne SPRITES, og en særlig egenskab er, at de bevæger sig i hvert sit plan, så de forskellige figurer kan bevæge sig ind over (eller bag ved) hinanden - medmindre man da har valgt at aktivere den indbyggede kollisionsdetektor.

Det er ikke svært at forestille sig de utal af rumskibe, der vil blive fremstillet med SPRITES, men der vil helt givet også være mere alvorlige opgaver, der kan løses med den nye grafik.

Hvor VIC-brugerne har været vant til at arbejde med otte farver på tegnene, kan der på Commodore 64 vælges mellem 16 forskellige kulører.

Hukommelsen

Når der defineres SPRITES, vil det naturligvis lægge beslag på en del af hukommelsen - og selv om der er 38 K til rådighed for BASIC-programmering, kan man nok forestille sig problemer. Derfor er det ganske snedigt, at indtil tre SPRITES kan gemmes i kassettebufferen, så man ikke optager lagerplads. Forudsætningen er naturligvis, at kassetten ikke anvendes så længe.

HI-FI lyd

Lyden er et helt kapitel for sig. Sammenligner man Commodore 64 med VIC-20, vil man blive overrasket over det teknologiske tigerspring, der her er taget. Tonerne kan tildeles forskellige karakteristikker helt efter ønske: Savtak, firkant eller trekant - og på firkant-tonerne kan man oven i købet ændre pulsbredden.

Det er endvidere muligt at bestemme, hvordan tonerne skal slås an, hvor hurtigt de skal falde til det såkaldte sustain-level og hvor længe, de skal være om at klinge helt ud. Det er faktisk en meget alsidig synthesizer, der er bygget ind i Commodore 64.



Ydre enheder

Til Commodore 64 kan man uden videre anvende printer, kassettebåndoptager og joysticks (der er plads til to på én gang), og VIC-diskettestationerne kan også benyttes, hvis der udskiftes en ROM i diskettestationen.

Derimod kan indstiksmoduler til VIC-20 ikke benyttes i forbindelse med Commodore 64. Programmer på ROM-kapsler eller hukommelsesudvidelser fra VIC-20 kan man altså ikke overføre til den større maskine.

VIC-brugere, der får den tanke at skifte deres nuværende maskine ud med en 64'er, vil naturligt bekymre sig om, hvorvidt eksisterende programmer, man har udviklet til VIC, kan anvendes i Commodore 64.

Det kan de, men hvis der i programmerne forekommer PEEK og POKE kommandoer eller, der er tale om programmer i maskinkode, må adresserne ændres, så de passer til den nye maskine.

VI & VIC

VI & VIC udgives af commodore data a/s,
Bjerrevej 67, 8700 Horsens.
Redaktion og tilrettelæggelse:
Baghuset Grafik/Design, Vejle.
Ansvarshavende redaktør: Steen Venbjerg.
Annoncer: Lise Tylvad,
Æblehaven 7, 7100 Vejle. Tlf. (05) 82 52 97
Tryk: Baghuset Grafik/Design, Vejle.
Redaktion og ekspedition:
Kornager 29, 7100 Vejle.
Giro nr. 6 17 39 93

ISSN 0108-2213

Streng-variablerne særlig vanskelige i overlay-program

Programmer, som kædes sammen ved hjælp af LOAD-kommandoer inde i programmerne, kan overføre variabler fra det ene program til det andet, men streng-variablerne giver specielle problemer. Men heldigvis er det problemer, som kan løses, når man ved, de eksisterer.

Oplysninger om alle variabler findes i en tabel i computerens lager, og der er sat syv bytes af til hver variabel - men da en streng-variabel kan fylde indtil 255 tegn, er der ikke plads i denne tabel. I stedet gemmes oplysning om, hvor lang strengen er samt om, hvor den i øvrigt er gemt.

For at spare plads kopieres strenge, der står i programmet IKKE ind i streng-arealet. Det vil sige, at hvis der f.eks. i et program står:

10 A$ = "ABCD"

så kopieres strengen ikke ind i streng-arealet, men i variabel-tabellen sættes pointeren til at pege på det sted **i programmet**, hvor variablen er defineret.

Det er i sig selv ganske snedigt, men det siger sig selv, at det vil gå galt, hvis man skifter programmet i computeren ud, men beholder variabel-tabellen, som peger et helt vildt sted ind i det nye program.

Men opgiv ikke. Man kan nemlig tvinge VIC til at kopiere strenge fra programmet op i streng-arealet. Strenge, der består af en sammensætning, vil nemlig altid blive kopieret ind i streng-arealet, og løsningen består derfor i at lave vores strenge om til sammensætninger. Man kan lettest sætte dem sammen med en tom streng, så programlinien kommer til at se således ud:

10 A$ = "ABCD" + ""

Kommer strengen fra en DATA-sætning, må man tage tilsvarende forholdsregler og skrive:

10 READ A$: A$ = A$ + ""

OVERLAY kæder mange programmer sammen - og passer man på, kan man godt undgå grå hår i hovedet

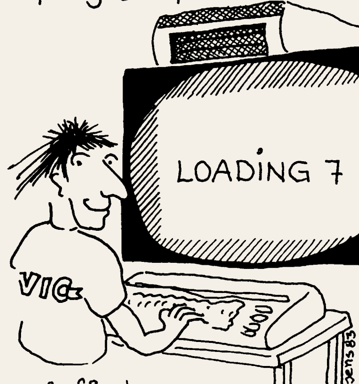
En af VIC's begrænsninger er den lagerplads, der er til rådighed. Det kan naturligvis løses ved at anskaffe udvidelsesmoduler, men her er en anden teknik, der både kan benyttes af den VIC-bruger, der ikke ønsker at købe flere RAM-moduler, eller som allerede har gjort det - og alligevel er i bekneb for plads.
Overlay er navnet på en teknik, der benyttes når flere programmer skal kædes sammen. Det ene program kan selv kalde det andet og starte det, evt. med alle variabler intakte. Bruger man diskettestation, kan programmerne kalde hinanden igen og igen, men har man kun en båndoptager til rådighed, kan programmerne kun afvikles i den rækkefølge, de er indspillet på båndet, men det byder også på mange muligheder.

LOAD

Forudsætningen for, at det kan lade sig gøre at lade ét program kalde et andet, er at kommandoen LOAD virker forskelligt alt efter, hvordan det anvendes. Normalt vil LOAD føre til, at det søgte program læses ind i computeren, og samtidig sættes de pointere, der viser, hvor programmet slutter og variabel-området starter. Programmet startes ikke af en direkte LOAD-kommando.
Forekommer LOAD derimod inde i et program, startes programmet så snart det er indlæst - og de nævnte pointere ændres ikke.

Variablerne

Det mest iøjnefaldende problem ved overlay-teknikken er at bevare de variabler, der



findes i computeren fra det ene program til det næste, og her gælder det netop om at sikre sig kontrol over pointerne, der holder rede på, hvor variablerne ligger. Det er let at lave et lille eksperiment, der afslører problemet: At et lille program kan kaldes fra et stort, uden at der sker noget, men at variablerne går tabt, når et stort program kaldes fra et lille.
Lav f.eks. et program som dette:

```
10 PRINT "HER ER
   PROGRAM ET"
20 INPUT A
30 REM DETTE
   PROGRAM ER DET
   LÆNGSTE
40 LOAD
   "PROGRAM TO"
```

Og det andet program:

```
10 PRINT "HER ER
   PROGRAM TO"
20 PRINT A
30 REM DETTE
   PROGRAM ER DET
   KORTESTE
```

Lagres de to programmer på diskette (eller efter hinanden på kassette), og man derefter indlæser og kører program et, vil man slutte med at have teksten "HER ER PROGRAM TO" stående på skærmen - og det tal, man har indtastet i program et, udlæses af program to.
Fjern derefter linie 10 og 30 i det første program, så det bliver det mindste. Nu vil program to udlæse et nul i stedet for den værdi, der blev indtastet i program et. Man har kort sagt knust sine variabler.

Løsningen

Problemet kan løses ved at gribe ind i pointernes værdier, så man »bilder« VIC ind, at ingen programmer er større end det første - eller med andre ord: Man giver VIC indtryk af, at det første program er større, end det er i virkeligheden.
Når man har skrevet (og lagret) alle de programmer, der skal bruges i overlay, finder man ud af, hvilket der er det største ved at benytte kommandoen PRINT FRE(0) i forbindelse med hvert enkelt program. Det program, som får det laveste tal frem på skærmen, er det længste, og herefter skal man finde ud af, hvor pointerne til start af variabel-området viser hen.
Skriv på skærmen:
PRINT PEEK(45), PEEK(46) og notér de to tal.
Første linie i det første program, der skal bruges i overlay skal herefter se således ud:

```
1 POKE 45, (værdi 1):
POKE 46,(værdi 2).
```

Kold start

Når man har indlæst et program, indeholder lagerpladserne 174 og 175 slutadresser + 1 på dette program, og når LOAD-kommandoen er givet direkte, kopieres disse værdier ind i pladserne 45 og 46.
Denne kopiering finder IKKE sted, når LOAD-kommandoen er givet i et program. Hvis programmet skal have en kold start, indledes det i overlay med følgende linier:

```
1 IF PEEK (45) = PEEK(47)
AND PEEK(46) =
PEEK(48) THEN 3
2 POKE 45,PEEK(174):
POKE 46,PEEK(175):CLR
3 REM PROGRAM-
START.
```

8 VIC bruger samme udstyr

VIC-Switch er navnet på en ny tilbehørsdel til VIC. Med VIC-Switch er det muligt at koble otte VIC-20 computere til én og samme diskettestation, og der kan også tilkobles printer til systemet.
Den mest iøjnefaldende anvendelsesmulighed for VIC-Switch er nok skoler og tilsvarende steder, hvor mange VIC-computere er i gang på samme tid.
Prisen for VIC-Switch er 1600 kr.

Oprettelse af relative filer

Der var fejl i program-eksemplet om oprettelse af relative filer i VI & VIC nr. 5. Det drejer sig om linie 20, hvor den korrekte programlinie skal se således ud:

20 OPEN 2,8,2,''RELFIL,L,'' + CHR$(20)

Hele oprettelsesproceduren komme så til at tage sig således ud:

```
10 OPEN 15,8,15
20 OPEN 2,8,2,"RELFIL,L,"+CHR$(20)
25 GOSUB 130
30 PRINT#15,"P"+CHR$(2)+CHR$(100)+CHR$(0)+CHR$(1)
50 GOSUB 130
80 PRINT#2,CHR$(255);
90 GOSUB 130
100 CLOSE 2
110 CLOSE 15
120 END
130 INPUT#15,EN,EM$,ET,ES
140 IF EN<20 THEN RETURN
150 IF EN=50 THEN RETURN
160 STOP
```

Linie 10
- åbner til fejl- og kommandokanalen. Kanalen bruges til at sende den kommando, som finder frem til en post. Fejl- og kommandokanalen bruges også til at undersøge, om der er opstået fejl ved læsningen eller skrivningen på disketten.

Linie 20
- opret en relativ fil:
OPEN 2,8,2,''RELFIL,L,'' + CHR$(20) = relativ fil med postlængden 20.

Linie 30-50-70
- undersøger, om der er opstået fejl. Det er klogt altid at se, om en diskoperation er gået godt!

Linie 40
- sætter postpointeren på det specificerede postnr. ''P'' sætter pointer (gennem fejl- og kommandokanalen).
CHR$(2)
- giver det logiske filnummer i linie 20 (det første af 2-tallene).
CHR$(100) og CHR$(0)

- sætter h.h.v. mindst betydende byte i postnr. og mest betydende byte i postnr. (low-byte og high-byte). Op til 255 poster kan være i low-byte (high-byte = 0).
Eksempel:
Postnr. 256 ser således ud:
CHR$(0) + CHR$(1)
Postnr. 258 ser således ud:
CHR$(2) + CHR$(1)
2 + 256 = 258
Sidste CHR$(1)
- angiver hvor i posten, man ønsker at starte. 1 er første position i posten.

Linie 60
- postnr. 100 tildeles værdien CHR$(255) i første byte, idet postpointeren er sat på post nr. 100 i linie 40. Systemet gør selv plads til de mellemliggende poster (1-99).

Linie 80-90
- lukker h.h.v.: RELFIL og fejl- og kommandokanalen.

Linie 110-130
- undersøger om operationerne er gået godt.

Fejl i brugervejledning

Ved trykningen af brugervejledningen til VIC-20 har der indsneget sig nogle mindre fejl. De er rettet på et rettelsesblad i nye brugervejledninger, og VIC-forhandlerne kan give »gamle« brugere en kopi af rettelsesbladet.

Lad VI & VIC hjælpe dig

VI & VIC skal være med til at du kan få mere fornøjelse af din VIC-20, og derfor indbyder vi dig til at komme med dine praktiske eller mere teoretiske spørgsmål.
Pladsen vil naturligvis sætte grænserne for, hvor mange spørgsmål, der kan blive besvaret her i bladet, men vi vil naturligvis først og fremmest lægge vægt på at besvare de spørgsmål, som optager mange VIC-brugere.

Skriv til VI & VIC på adressen Kornager 29, 7100 Vejle, hvis du har spørgsmål, du gerne vil have behandlet.

Her er programmet for beregning af bedste kombination af frimærker

Det **kan** lade sig gøre at lave et VIC-program, der finder frem til den optimale kombination af de forhåndenværende frimærker, når et brev skal frankeres. Her er i hvert fald et program, som løser problemet - men det udløser ikke den præmie, vi udlovede i sidste nummer af VI & VIC, for vi har faktisk ikke modtaget én eneste løsning på problemet.

Den gamle redaktør frygtede, han havde stillet en uløselig opgave og at de stakkels VIC-brugere sad rundt om i landet med hovedpine efter at have forsøgt sig med frankeringsproblemet - og så gik han i stedet i gang med selv at knække nøden.

Og faktisk er det ikke så vanskeligt. Ved hjælp af et antal NEXT . . . FOR løkker gennemgår programmet simpelt hen samtlige mulige kombinationer, og hver gang der findes en kombination, der resulterer i en mindre overfrankering end den tidligere, gemmes den som den »hidtil bedste«. Det samme sker, hvis overfrankeringen er den samme, men antallet af anvendte frimærker er mindre.

I det program, der gengives her på siden, er der mulighed for at arbejde med seks forskellige frimærker, men det vil være en simpel sag at udvide det.

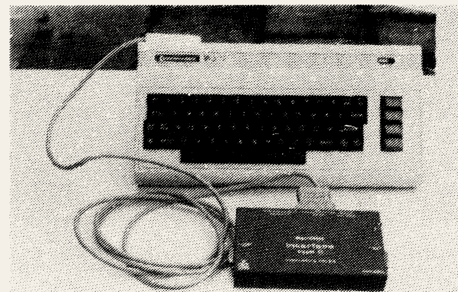
Det største problem med programmet er, at jo mindre værdier der findes at arbejde med, jo større bliver antallet af kombinationsmuligheder - og man havner meget let i en situation, hvor det næsten er hurtigere selv at løbe ud med brevet end at vente på, at VIC har regnet færdig.

Har man f.eks. 10-øres, 50-øres og 1 kr. frimærker i systemet, tager det VIC fire og et halvt minut at regne ud, hvilken kombination, der er bedst, hvis man skal sætte 5 kr. på et brev.

Programmet gennemgår nemlig alle muligheder, og det vil i dette eksempel sige 2.500 kombinationsmuligheder. Vi tør slet ikke tænke på konsekvenserne ved endnu flere forskellige (små) frimærkeværdier.

```
10 ST$="    ****************":US$="────────────────────"
20 PRINT"□▓▓▓▓▓▓"ST$:PRINT"▓      VI & VIC'S":PRINT"▓      PORTOPROGRAM▓":PRINTST
$
30 FORT=1TO3000:NEXT:GOTO4000
1000 REM ** KR./ØRE**
1001 PR=INT(PR*100+.5)/100:PR$=STR$(PR):IFPR<1THENPR$="0"+MID$(PR$,2)
1002 IFPR-INT(PR)=0THENPR$=PR$+"."
1003 IFASC(RIGHT$(PR$,3))<>46THENPR$=PR$+"0":GOTO1003
1004 PRINTTAB(21-LEN(PR$))PR$:RETURN
1200 GETS$:IFS$=""THEN1200
1210 RETURN
1300 POKE36878,15:POKE36876,200:FORV=1TO20:NEXT:POKE36876,0:RETURN
1400 POKE36878,15:POKE36875,177:FORV=1TO200:NEXT:POKE36875,0:RETURN
1500 PRINT"▓▓    *** SIKKER? ***"
1510 GETT$:IFT$=""THEN1510
1520 PRINT"□                          ":RETURN
4000 REM ***PORTO***
4010 PRINT"□       FRANKERING":PRINTUS$:PRINT"FRIMAERKER (ØRE):▓▓"
4020 FORT=1TO6:IFM(T)=0THENINPUT"BELØB";M(T)
4030 IFM(T)=0THENAM=T-1:GOTO4040
4035 NEXT:AM=6
4040 PRINT"▓":INPUT"PORTO";P
4050 REM***MAX.ANTAL***
4060 FORT=1TOAM:MM(T)=INT(P/M(T)):IFMM(T)<>P/M(T)THENMM(T)=MM(T)+1
4070 NEXT
5000 OP=M(1)*MM(1)
5001 AN=0:FORT=1TO7:IFMM(T)>ANTHENAN=MM(T):NEXT
5010 FORA=MM(1)TO0STEP-1
5020 FORB=MM(2)TO0STEP-1
5030 FORC=MM(3)TO0STEP-1
5040 FORD=MM(4)TO0STEP-1
5050 FORE=MM(5)TO0STEP-1
5060 FORF=MM(6)TO0STEP-1
5501 AF=A+B+C+D+E+F
5502 FP=A*M(1)+B*M(2)+C*M(3)+D*M(4)+E*M(5)+F*M(6):IFFP>OPORFP<PTHEN5990
5503 IFOP=FPANDAF>ANTHEN5990
5505 AN=AF:OP=FP:GOSUB1400:PRINT"□      FRANKERING":PRINTUS$
5506 PRINT"STK.  ØRE        I ALT ────  ───       ────"
5510 IFA>0THENPRINTATAB(5)M(1);:PR=A*M(1)/100:GOSUB1000
5520 IFB>0THENPRINTBTAB(5)M(2);:PR=B*M(2)/100:GOSUB1000
5530 IFC>0THENPRINTCTAB(5)M(3);:PR=C*M(3)/100:GOSUB1000
5540 IFD>0THENPRINTDTAB(5)M(4);:PR=D*M(4)/100:GOSUB1000
5550 IFE>0THENPRINTETAB(5)M(5);:PR=E*M(5)/100:GOSUB1000
5560 IFF>0THENPRINTFTAB(5)M(6);:PR=F*M(6)/100:GOSUB1000
5570 PRINTTAB(16)"─────"
5600 PRINT"PORTO ANVENDT";:PR=OP/100:GOSUB1000:PRINT"NØDVENDIGT";:PR=P/100:GOSUB
1000
5602 PRINTTAB(16)"====="
5603 PRINT"▓OVERFRANKERING";:PR=(OP-P)/100:GOSUB1000
5604 :IFOP-P=0THENGOSUB1300
5605 PRINTTAB(16)"====="
5990 NEXT:NEXT:NEXT:NEXT:NEXT:NEXT:PRINT"▓▓  ***    FLERE?    ***":FORT=1TO5:GOSUB
1300:NEXT
5991 GETS$:IFS$=""THEN5991
5992 IFS$="N"THENGOSUB1500
5993 IFT$="J"THENEND
5994 GOTO4040
```

VIC-20 hjertet i avanceret demonstrationssystem for radio- og hifi-branchen



Myten om, at de datamater, der normalt karakteriseres som hjemmedatamater, ikke kan bruges i professionel sammenhæng, har udsigt til at blive aflivet.
I hvert fald har et lille elektronikfirma i landets brogede vifte af nicheproduktioner taget VIC-20 til sig i forbindelse med salg af demonstrationsudstyr til radio- og hifibranchen.

Interface

Det drejer sig om Lohse Electronics i Måløv vest for København, der gennem firmaets otte år lange levetid blandt andet har udviklet systemer til demonstration af i første omgang højttalere og sidst videoanlæg. Først var der blot tale om en række komponenter til at styre præsentationen af højttalere. Siden er der koblet en VIC-20 på, der har til opgave at vise den pågældende højttalers specifikationer og data på skærmen i kraft af et specialudviklet interface. Kunden har naturligvis mulighed for at få de viste data med sig hjem i form af en udskrift.

Da firmaet i sin produktudvikling nåede dertil, at man ville have systemet understøttet af en datamat, faldt valget på VIC-20. Dels på grund af prisen og dels på grund af, at denne datamats datakraft og anvendelsesmuligheder også egner sig til brug i denne sammenhæng, fortæller firmaets daglige leder og indehaver Bjarne Lohse.

Færdig løsning

Den samlede løsning hedder Digi-Demo og omfatter omskiftersystem, interfacebox, det færdige program og naturligvis VIC-20'eren. Bjarne Lohse forklarer, at netop VIC-20 har en pris, der står i et rimeligt forhold til hele produktet. En anden type datamat ville fordyre løsningen urimeligt meget, udtaler Bjarne Lohse, der også oplyser, at selve programmet koster 200 kr. Der lægges vægt på, at de specialdesignede programmer, der på traditionel vis er skrevet i Basic, er helt færdigudviklede, når de sælges, da man ikke skal lægge an på, at brugeren selv kan programmere.
Der er solgt omkring 100 såkaldte audio systemer til dato herhjemme, men nu efter at datamaten er koblet på, forventer man en øget omsætning især på eksportmarkedet. I forvejen ligger hovedparten af firmaets omsætning, hvilket vil sige godt en million kr., netop på eksport.
Dog er det første system med videoomskifter i den sidst udviklede udgave allerede solgt til en radioforhandler på Amager.



Solgt i Schweiz

Det, Lohse Electronics nu har gjort, er at udvikle det interface, der forbinder det eksisterende audiosystem med VIC-20 datamaten. Og der vil også blive udviklet et interface til video-præsentation, hvor TV-displayet skiftevis kan vise forskellige videoanlægs billedkvalitet og disses specifikationer i en tovejs-kommunikation mellem interface og datamat. Den nye videreudviklede videoomskifter er allerede solgt til blandt andet Schweiz, hvorfra firmaet har fået gode tilbagemeldinger, oplyses det. Totalt set er der på nuværende tidspunkt solgt 25 systemer med interface til udlandet.
Bjarne Lohse oplyser, at man fremover også vil satse på andre brancher, for eksempel til demonstration af modetøj.

En styrke hos VIC er, at den kan programmeres både i BASIC og i maskinkode

I sidste nummer af VI & VIC så vi på den aritmetiske enhed i selve mikroprocessoren. Der er stadig et antal afdelinger i mikroprocessoren, som vi skal have gennemgået for at få den fulde forståelse af maskinkode. Det er ting som status register, flag, hop og spring, stakregister, indexregistre og interput. Så der er nok at tage fat på.
For at undgå, at artiklen skal udvikle sig til en omgang tør tekst, har vi valgt at foregribe tingenes gang en smule. Vi briger derfor i dette nummer, de første programeksempler, der direkte kan afprøves på en VIC-20. Ovenfor nævnte ting og yderligere eksempler bringes så i de næste numre.

VIC-fordel

VIC-20 har en fordel frem for mange andre mikrocomputere i, at den kan programmeres i både BASIC og maskinkode, hvilket gør det muligt for programmøren at kombinere BASIC med subrutiner i maskinkode. VIC arbejder normalt i BASIC, og der findes seks måder at få kontakt med maskinkoden på. De to første anvender BASIC kommandoer, det er USR(X) og SYS. Begge kommandoer starter en maskinkode, hvis startadresse enten er angivet i en side nul plads, eller i kommandoen selv. De næste fire metoder går ud på at tilføje maskinekode rutiner til operativsystemet.
1 - BASIC kommandoen USR(X) overfører kommandoen til en rutine, hvis adresse skal gemmes i lokation 1 og 2, før kommandoen udføres. Parameteren X er en værdi, som selve rutinen kan bruge. Værdien beregnes og placeres i computerens floating point akkumulator no. 1, i lokation 61. (Computerens mellemlager, når den laver matematiske beregninger). Det kan også lade sig gøre for maskinkode rutinen at returnere en værdi den anden vej. Værdien skal blot placeres i akkumulatoren i det rigtige format, så vil værdien blive overført til variablen.
2 - BASIC kommandoen SYS (X) får computeren til at gå til adresse X, og fortsætte i maskinkode derfra. X er enten en variabel eller en konstant, med en værdi der svarer til maskinkodens start. Parametre kan udveksles mellem BASIC og maskinkode ved hjælp af PEEK og POKE kommandoer, der kan placere eller læse værdier til og fra hukommelseslokationer.
3 - Hvis maskinkode rutinerne er placeret i ROM eller ESROM, og starter i hukommelseslokation $AOOO, så gør VIC's operativsystem det muligt at hoppe til denne rutine, når VIC tilsluttes, i stedet for den normale opstart. Det er meget anvendeligt, idet det gør det muligt at ændre VIC's operativsystem. Det kan være enten for at tilføje nye ekstra kommandoer, eller for at ændre de bestående, eller simpelthen for at omgå BASIC, og erstatte det med speciel skrevet software. (Dette er meget normalt i f.eks. spilmoduler).
4 - Et program kan også indsættes i interrupt rutinen, der udføres hver 60. sekund. Herved kan man f.eks. se på I/O portene for et input signal, eller individuelt sætte visse taster ud af funktion. Enhver situation, hvor det er nødvendigt at have et program kørende sideløbende med et andet, vil kunne udnytte denne metode.
5 - Denne metode går ud på at indsætte ekstra kode i CHRGOT rutinen. Denne rutine bruges af BASIC til at hente hver BASIC linie ind til undersøgelse før kørsel. Ved at undersøge hver BASIC linie, før den udføres, kan man tilføje nye BASIC kommandoer, der udføres af egen maskinkode. Vi vil senere se på, hvorledes dette gøres i praksis.
6 - RAM vektor adresse tabellen kan også bruges til at indsætte eller helt udskifte BASIC eller system rutiner. Dette er identisk med metode 4, og kunne bruges til at ændre interruptliniernes funktion eller ændre I/O rutinerne.

Skrivning af maskinkode

Det at skulle skrive selv et lille og simpelt maskinkode program, kan forekomme ret afskrækkende, men hvis man anvender en rigtig og systematisk fremgangsmåde, behøver det ikke at være svært.
Et maskinkode program skrives helt forskelligt fra et BASIC program. Hvor man ret let kan sammenstrikke et BASIC program, og så senere pudse det af, ved at rette, slette og indsætte linier, så skal et maskinkode program skrives i sin endelige version. Senere ændringer kan for det meste kun opnås ved, at hele programmet omskrives på ny. Maskinkode er, til forskel for BASIC, meget afhængig af kodernes placering i de enkelte lokationer. Tilføjelse af et par instruktioner midt i programmet, nødvendiggør næsten altid, at en mængde forgrenings- og hopadresser skal rettes, og flyttes. Maskinkode kræver også, at programmøren er mere opmærksom på små detaljer, så som flag status o.lign., så programmerne må planlægges meget nøje, før de skrives. Gør man ikke det, vil udviklingen af et maskinkode program kræve langt større arbejde og besvær, end egentlig nødvendigt, og det vil også udstyre det endelige resultat med en stor sandsynlighed for fejl. Det anbefales på det kraftigste, at du, før du selv går igang med programmeringen, studerer andre 6502 maskinkode rutiner. Prøv at finde ud af, hvorfor koden er skrevet, som den er, og hvad den gør.

Et eksempel på program skervet i maskinkode:

Alle tegn på skærmen i ét nu

Første trin i planlægningen af et maskinekode program er, at definere hvad programmet egentlig skal kunne gøre, og derefter yderligere at specificere de enkelte trin. Prøv som en demonstration at studere det følgende eksempel, der kan vise samtlige ASCII-tegn på skærmen.

Først sættes skærmpositions pointeren LOK til begyndelsen af skærmen, adresse 32768 - sæt ASCII-tegn værdien TEGN til nul - gem kode TEGN i lokation LOK - læg én til LOK - læg en til TEGN - hvis TEGN er større end 255, har alle tegn været vist, og programmet stopper, hvis ikke, skal programmet gå tilbage og vise næste tegn.

Udfra beskrivelsen har vi bestemt, at vi skal bruge to variable, LOK og TEGN, og vi kan også se, at programstrukturen er en løkke, med en tekst. For et kort program som dette eksempel, er en skriftlig beskrivelse egentlig ikke nødvendig, da man sagtens kan huske, hvad det er,

```
10 REM ******************************
20 REM * BASIC LOADER FOR MASKINKODE
30 REM * EKSEMPLET VISER DISPLAY
40 REM ******************************
100 DATA 832 : ** KODENS START I DECIMAL **
105 REM ** MASKINKODEN I HEXADECIMAL **
110 DATA A9,FF,8D,40,03,A2,FF
115 DATA AD,40,03,9D,00,1E,A9
120 DATA 03,9D,00,96,CE,40,03
125 DATA CA,D0,EF,60
9000 DATA * :REM ** SLUT PA KODE **
9005 REM ** FØLGENDE LINIER ER **
9006 REM ** SELVE LOADER PRG.  **
9010 READ L
9020 READ A$
9030 C=LEN(A$)
9040 IF A$="*"THEN9140
9050 IFC<1ORC>2THEN9130
9060 A=ASC(A$)-48
9070 B=ASC(RIGHT$(A$,1))-48
9080 N=B+7*(B>9)-(C=2)*(16*(A+7*(A>9)))
9090 IFN<0ORN>255THEN9130
9100 POKEL,N
9110 L=L+1
9120 GOTO9020
9130 PRINT"BYTE"L"=("A$") ????"
9140 END
```

man vil have programmet til at gøre. Men for længere programmer, er det en uomgængelig del af programudviklingen. Fra den skriftlige beskrivelse kan man konstruere et rutediagram, som eksemplet i FIG. 2. Rutediagrammet kan betragtes som en billedlig version af den skriftlige beskrivelse, og er derfor simplere at følge.

Del programmet

Ved lange programmer kan rutediagrammet og den skriftlige beskrivelse blive meget indviklet og kompliceret. Så det er derfor en god idé at dele den op i mange små dele. Hver del kan så være enselvstændig rutine eller subrutine. Hver del betragtes så som en selvstændig del, hvilket gør skrivningen og fejlfindingen meget lettere. Rutediagrammet viser den logiske gang igennem programmet, og de fleste logiske fejl, og tanketorske kan for det meste allerede fjernes på dette stade, hvilket sparer en mængde programmeringstid.

Når man har tegnet rutediagrammet, er det næste trin at opbygge en tabel af systemvariable og lokationer, som rutinen skal bruge. I eksemplet skal vi ikke bruge nogle systemrutiner, men vi har brug for to variable:

LOK - pointer til positionen på skærmen, hvor tegnet skal skrives.

TEGN - værdien på det ASCII tegn, der skal skrives. Det er vigtigt, at tabellen indeholder alle de værdier, der skal bruges, da der skal reserveres den nøjagtige plads til dem, når programmet skrives.

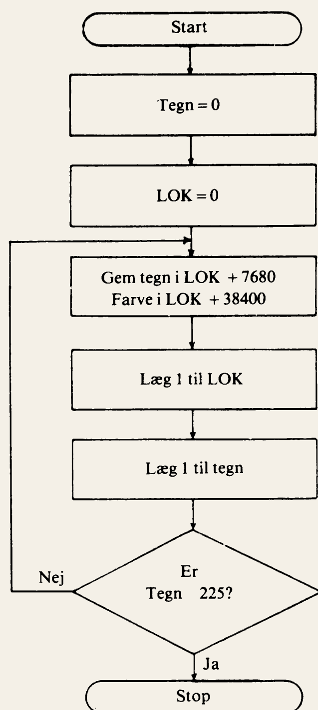


FIG. 2.

Selve skrivningen

Når vi har defineret programmets logiske forløb, de nødvendige variable, og eventuelt de nødvendige systemrutiner, kan vi begynde at skrive vort maskinkode program. Det er formodentlig det bedste at tegne en udvidet version af rutediagrammet. Her nedbryder vi hver enkelt logisk trin til et antal undertrin, hver svarende til en maskinkode instruktion. Bemærk i FIG. 3, at variablen LOK nu gemmes som indholdet af X-indeks registeret. Indekseret adressering er den bedste programmeringsmåde, når man skal placere data i på hinanden følgende lokationer. Se også, at X-indeks registret (altså LOK), starter med værdien 255, og tælles nedad, istedet for nul, og tælles opad, som vist i det oprindelige rutediagram. Det er gjort, fordi det er lettere at teste for nul end 255. Bemærk også, at programmets opbygning ikke nødvendigvis er den mest optimale med hensyn til hastighed og plads, men alene tjener overskuelighdens og eksemplets formål.

Tre måder

Det næste trin er at skrive de maskinkode instruktioner, der passer til hver enkelt trin. Dette kan gøres på tre forskellige måder, helt afhængig af hvilket udstyr, man har til rådighed, og programmets størrelse. Den første metode er at skrive koden i hånden, på en formular, idet man benytter sig af instruktionsforkortelserne eller operand adresse/værdier. Se FIG. 4. Maskinkode, der er skrevet på denne måde, indlæses lettest i computeren med et BASIC loader program.
Denne metode går ud på at skrive koden i mnemonicform (forkortelsesform), og så indtaste koden v.hj. af assembleren i maskinekode monitormodulet. Endelig, den tredie metode, der er mest anvendelig for meget lange maskinkode programmer, er at anvende en CBM computer og det tilhørende Assembler Development System, (Maskinkode udviklings System). Her indtastes koden i en editor, ligesom BASIC, og kan rettes. Den gemmes på diskette, hvorfra et assemblerprogram kan oversætte teksten til egentlig maskinkode, der så kan læses ind i computeren til udførelse.

Kodningsformular

Hvis man håndprogrammerer et program, anbefales det at benytte sig af en kodningsformular, som vist i eksemplet. Det nedsætter antallet af fejl, der kan gøres på dette stade. På formularens første side skrives en liste over samtlige variable, I/O lokationer og systemrutiner. Hver variabel tillægges så meget plads, som den vil fylde i computeren. De fleste vil være enkelt byte variable, men nogle kan fylde to eller tre lokationer, og i det tilfælde, at det drejer sig om tekst, kan det være nødvendigt at reservere et større dataområde. Når man gemmer en variabel, der fylder mere end én byte, et det en god idé at gemme tallene i en fastlagt rækkefølge, med den mindst betydende byte først, og den mest betydende byte sidst. På denne måde er det lettere at finde ud af, hvilken del af en variabel, man arbejder med. På denne måde kan indeksregistre også bruges til at hente en række variable til processoren i den rækkefølge, som de naturligt skal beregnes.

Plads i side nul

Programvariable kan gemmes i enhver del af RAMmen, der ikke i forvejen optages af programmer eller systemvariable. For at opnå den størst mulige hastighed, bør et programs variable gemmes i computerens nul-side, de første 255 bytes. På Vic er denne side næsten optaget af systemvariable, men der er dog nogle få, derkan bruges, hvis de vælges med omhu. Hvis BASIC ikke skal benyttes, er hele den del af side nul, som BASIC benytter, (lokation 0 til 143) fri til brug. Den resterende del af side nul anvendes af operativsystemet og kan måske benyttes af maskinkode programmet. Hvis både BASIC og maskinkode skal benyttes i samme program, er antallet af ledige side nul lokationer ret begrænset. (Lokationerne 87 til 96 er bedst). Hvis man har brug for større dele af side nul, bør de eksisterende data flyttes til et beskyttet område før brug, og flyttes tilbage, når maskinkoderutinen er færdig.

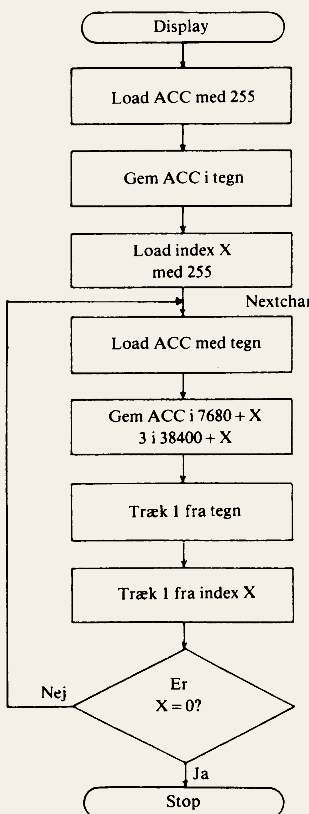


FIG. 3.

Tælle bytes

Med det udvidede rutediagram som udgangspunkt, kan man nu gå igang med at skrive maskinkoden på formularen ved hjælp af instruktionsforkortelserne (mnemonics). Først skrives kodens start lokation i adressekolonnen. Instruktionens adresseringsmåde skal også skrives i den rette kolonne. Dette er vigtigt, idet man skal kunne beregne, hvor mange byte instruktionen skal bruge for at beregne i hvilken linie, dvs. hvilken adresse, den næste instruktion skal begynde i. Label kolonnen skal kun benyttes, hvis den linie er begyndelsen på en underrutine, eller et mål for et hop eller spring. På rutediagrammet indikeres en label ved, at en position har mere end én indeller udgang. En label, (etiket), kan være ethvert navn, men det er bedst at bruge et ord, der er beskrivende for rutinen. I eksemplet er begyndelsen kaldt for DISPLAY, og starten på løkken kaldes NEXTCHAR., (Næste tegn). Operand kolonnen benyttes kun af instruktioner, der refererer til andre lokationer i programmet, og vil i så fald være en label eller variabel. Efterhånden som maskinkoden indtastes, bør man også udfylde bemærkningsfeltet. Enten med simple henvisninger til rutediagrammet eller en mere komplet beskrivelse. Kun på denne måde kan man være sikker på senere at kunne forstå programmets funktion.

Fortsættes side 10

Fortsat fra side 9

Find fejlene

Når programmet først er skrevet, skal det undersøges for logiske fejl, før indtastningen. Det er meget tidsbesparende, hvis fejlene findes før programmet indtastes. Indtastningen, eller »håndassembleringen«, foregår i to trin. Den første består i, ved hjælp af en liste over instruktionerne, at finde værdien for hver enkelt instruktion, med den specificerede adresseringsmåde. Denne hexadecimale værdi skrives i operationskode kolonnen på formularen, på samme linie som instruktionen. Hvis adresseringsmåden er andet end »implied« eller »absolut«, skal den eller de næste bytes bruges til at indeholde en adresse eller en værdi, specificeret i operand kolonne. Hvis adresseringsmåden er immidiate, skal operand kolonnen indeholde en hexadecimalværdi, der overføres til opkode kolonnen, på linien efter instruktionskoden.

Tre talsystemer

Det skal altid tydeligt fremgå, hvilket talsystem, der er anvendt. Der anvendes således et % for at angive et binært tal, et $ for at angive et hexadecimalt tal, og et decimalt tal, hvis der ikke er noget tegn foran tallet. Det er også fastlagt, at en instruktion i immediate mode efterfølges af et »dobbeltkryds« (hatch - havelåge). I alle andre adresseringsmåder skal symbolet i operand kolonnen henvise til enten en variabel eller en label. Hvis det er en variabel, kan variablens adresse hentes fra tabellen på formularens første side. Hvis instruktionen er en hop- eller forgreningsinstruktion, der vil overføre programudførslen til en anden del af programmet, skal operand kolonnen indeholde en label. Bemærk, at 6502 kræver, at alle adresser gemmes med den mindst betydende byte først, efterfulgt af den mest betydende byte. Adressen $0340 gemmes derfor som: 4003.

Forgreninger

Ved slutningen af første trin i assembleringsprocessen, skal operationskode kolonnen indeholde en liste af hexadecimale værdier, én for hver lokation i hukommelsen. Undtagelserne er hop-og forgreningsadresser, der beregnes i anden omgang. Hopadresser giver ikke noget problem, da de altid gemmes indirekte, eller mere normalt, som en absolut adresse. Indholdet i operationskode kolonnen kan derfor direkte hentes fra den relevante labels adresse. Forgreningsinstruktionerne benytter sig alle af relativ adressering, hvor forgreningen, der kan være fremadrettet eller bagudrettet, beregnes ud fra instruktionens placering, og ikke fra en fast plads i hukommelsen. Det er forskellen fra den nuværende lokation, til forgreningen, der kan være på til 127 bytes væk, fremad eller bagud, der skal beregnes af programmøren. Det kræver stor omhu, idet en forkert beregning kan få programmet til at gå helt skæv i byen. En fremadrettet forgrening beregnes ved at tælle antallet af bytes, fra instruktionen til placeringen af den label, der indgår i den, og trække to fra dette tal. Hvis forgreningen er bagudrettet, beregnes den ved at tælle antallet af bytes fra instruktionen til placeringen af den label, der indgår i den, lægge én til, og trække resultatet fra 255. Resultatet omregnes så til et hexadecimalt tal, og skrives i kolonnen efter forgreningsinstruktionen.

Når alle hopadresser er beregnet, og alle operationskoder er noteret som hexadecimale tal, kan programmet indtastes i computeren. Før dette gøres, er det klogt at undersøge programmet en ekstra gang, især er det vigtigt, at operationskoderne er korrekte. (Vær sikker på, at du kan se forskel på 8 og B, og R og 4). Først derefter indtastes koden i VIC, ved hjælp af en BASIC loader, eller maskinkode monitoren. Når programmet er indtastet, bør det gemmes øjeblikkeligt på kassette eller diskette, før det forsøges udført, da det er meget sjældent, at et maskinkode program kører helt fejlfrit første gang. Ved hjælp af monitoren kan hukommelsen så sammenlignes med formularen, for på den måde at finde eventuelle indtastningsfejl. Hvis sådanne findes, rettes de, og programmet gemmes igen før test. Så kan man prøve at udføre det.

Hvis der er en programmeringsfejl, vil den sandsynligvis få maskinen til at gå helt grassat. I så tilfælde må man slukke og tænde for computeren, og genindlæse den gamle version, hvorefter man kan forsøge at finde yderligere indtastningsfejl, eller logiske programfejl. Findes sådanne, må man korrigere dem og gennemgå den beskrevne procedure igen. De mest almindelige fejl er således: Indtastningsfejl, kodefejl og forkert beregnede forgreningsadresser.

På sporet af fejl

Den bedste måde at finde fejl på, er systematisk at arbejde sig gennem programmet, medens man indsætter en break (BRK) kommando på strategisk vigtige steder. Derved kan man undersøge variable og programtæller, på forskellige steder i programudførslen, for derved at få en idé om, hvornår programmet kører af sporet. Efter at de fatale fejl er fjernet, vil man måske konstatere, at programmet stadig ikke kører korrekt, og giver mærkelige resultater. Ikke-fatale fejl skyldes som regel, at man har misforstået programmets logiske forløb, eller overset et status flag, brugt de forkerte variable, og meget almindeligt: brugt den forkerte forgreningsinstruktion.

Effektiv maskinkode programmering er ikke svært, man skal blot følge en helt fastlagt metode, og hele tiden have øjnene åbne for de små detaljer. Plus en masse erfaring.

PROGRAM DISPLAY

DATE

PAGE 1

ADDRESS		OPCODE	LABEL	MNEMONIC	AD MODE	OPERAND	FLAGS						CYCLE	COMMENT
MSB	LSB						Z	N	C	I	D	V		
03	4 0	—	CHAR	—										VARIABLE FOR ASCII CHARACTER
	1	A9	DISPLAY	LDA	#	255								START – SET UP LOOP COUNT
	2	FF		/										AND CHARACTER VALUE
	3	8D		STA	ABS	CHAR								INITIALISE 'CHAR'
	4	40		/										
	5	03		/										
	6	A2		LDX	#	255								SET INDEX REG = 255
	7	FF		/										
	8	AD	NEXTCHAR	LDA	ABS	CHAR								GET 'CHAR'
	9	40		/										
	A	03		/										
	B	9D		STA	ABS,X	$1E00, X								STORE AT START OF VIDEO RAM
	C	00		/										+ INDEX
	D	1E		/										
	E	A9		LDA	#	03								SET COLOUR = RED
	F	03		/										
	5 0	9D		STA	ABS,X	$9600, X								STORE COLOUR IN COLOUR RAM
	1	00		/										AT $9600, X
	2	96		/										
	3	CE		DEC	ABS	CHAR								PUT NEXT ASCII CHARACTER
	4	40		/										IN 'CHAR'
	5	03		/										
	6	CA		DEX	IMP		x							POINT TO NEXT SCREEN
	7	D0		BNE	REL	NEXTCHAR								LOCATION – LAST CHARACTER?
	8	EF		/										
	9	60		RTS	IMP									END AND RETURN FROM SUBROUTINE
	A													
	B													
	C													
	D													
	E													
	F													

FIG. 4.

Der er tretten måder at finde frem til den rette adresse i VIC

Enhver instruktion i et maskinkode program indeholder information om, hvor i lageret den kode der skal arbejdes på, befinder sig. Den samme instruktion kan eksistere i mange forskellige former, afhængig af hvor i lageret data befinder sig. Hver af disse former, kaldes en adresserings måde. der er tretten forskellige adresserings måder, og de fleste instruktioner kan forekomme på mere end een måde.

LDR instruktionen kan således forekomme på otte forskellige måder, og seks måder, der er kombinationer af de normale og indexering.
Implied: Ingen
Akkumulator: Akkumulator
Immediate: Direkte
Absolute: Absolut
Zero page: Side nul
Relative: Relativ
Indirect: Indirekte
Absolute X Indexed: Absolut, indekseret med x register
Absolute Y Indexed: Absolut, indekseret med y register
Zero page X indexed: Side nul, indekseret med x register
Zero page Y indexed: Side nul, indekseret med y register
Indirect indexed: Indirekte, indekseret med y
Indexed indirect: Indirekte, indekseret med x
Den simpleste måde er Implied adressering, der udelukkende bruges af enkelt byte instruktioner, der berører processorens interne registre. I en instruktion som CLC (clear carry, slet mente), hentes der ingen data, hvorfor der ikke er brug for nogen adresse. Det er underforstået at det er et internt register, her status-registeret, der skal arbejdes på.

Akkumulator måden bruges af instruktioner der skal foretage logiske operationer på data i akkumulatoren. Denne måde er en version af Implied, og alle instruktioner fylder een byte.

Immediate, direkte-adressering, bruges altid når programmøren ønsker at udføre en operation, hvor der skal bruges en konstant. Hvis man f.eks. vil bruge værdien 25 i akkumulatoren, skal vi bruge LDA instruktionen i sin immediate form. I denne adresserings måde gemmes data i byten der følger umiddelbart efter instruktionen. (LDA $25).

Hverken immidiate- eller implied-måden bruger en adresse hvor der gemmes data, og er kun af lille værdi, når man arbejder med variabler. At adressere en eller anden plads i hukommelsen, kræver en fuld 16 bit-, eller to byte-, adresse gemt i instruktionens operanddel. Denne adresse peger så på den plads i hukommelsen, hvorpå man ønsker instruktionen udført, eller resultatet gemt. Denne adresseringsmåde kaldes for »absolut adressering«.

Forkortet form

Der kan bruges en forkortet form for absolut adressering, når pladsen man ønsker at bearbejde, befinder sig på side nul. Dette er det eneste sted hvor sideopdelingen, beskrevet tidligere, har nogen betydning for programmøren. Side nul er de første 256 hukommelsespladser i lageret. Dette kaldes for: side nul adressering, og der bruges kun een byte, til at pege på hukommelses pladsen. En sådan to byte instruktion er meget hurtigere end f.eks. absolut adressering, så det er derfor normalt at gemme alle variabler i side nul. Når man kører maskinkode programmer i VIC, bør man kun anvende de første 143 bytes i side nul, idet maskinen sandsynligvis vil gå i stå, hvis man søger at benytte resten.

En anden adresseringsmåde benyttes udelukkende af forgrenings- og hoppeinstruktioner. Den kaldes for: relativ adressering. I denne adresseringsmåde følges instruktionen af en enkelt operand. Denne byte specificerer ikke en hukommelsesplads, som i side-nul adressering, men afstanden fra denne instruktion frem til den instruktion, der skal udføres næste gang. Da denne afstand kan være både positiv, som negativ, bruges det ottende bit til at bestemme retningen, hvilket muliggør et hop 127 bytes frem, eller 128 tilbage i programmet.

Beregnet adresse

I nogle programmer kan det være nødvendigt at have en beregnet adresse istedet for en fast, som i absolut adressering. Dette gøres med indirekte adressering. Denne slags instruktioner følges blot af en enkelt byte, der peger på en adresse i side nul, der så, sammen med den umiddelbart følgende adresse, indeholder en værdi der angiver den egentlige adresse. Alle de indirekte adresseringsmåder er indexede, bortset fra JMP, hop, instruktionen.

Indekseret adressering benytter indholdet af et af de to indeksregistre som et offset, (forskydning), til den adresse der gemmes som operand.

Adressen der gemmes som operand, kan enten være en absolut to byte adresse, eller en byte side nul adresse. Derfor er der mulighed for ialt fire adresserings måder. Denne adresseringsmådes mest almindelige brug er, ved tilgang til flere på hinanden følgende hukommelsespladser, tabeller eller data.

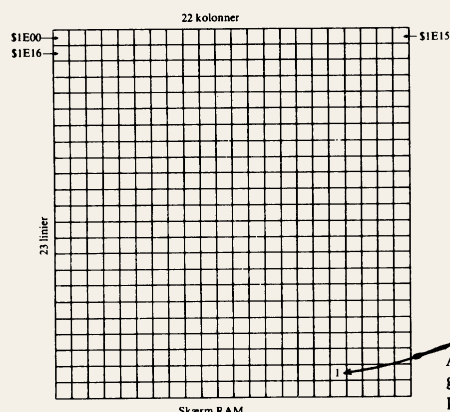
Resultat = pointer

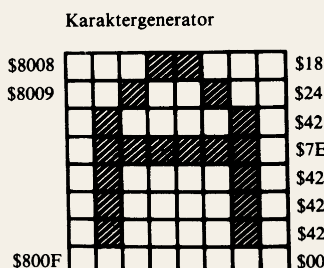
I indekseret indirekte adressering, adderes indekregister x, til operandens side nul, hvor den egentlige adresse gemmes. Det er så denne to byte adresse der bestemmer, hvor i hukommelsen den byte befinder sig der skal arbejdes på. Denne adresserings måde bruges overvejende til at hente data ud af en tabel eller adresseliste, f.eks. ved aftastning af I/O apparater, og streng operationer.
I indirekte indekseret adressering, hentes først de to byte i side nul, som operanden peger på, dernæst offsettes, adderes, denne to-byte adresse med indholdet af indeksregisteret, og man har den endelige adresse.
Indirekte indekseret adressering kombiner fordelen af en pointer der kan pege på et hvilket som helst sted lageret, med indeksregisterets offset mulighed. Det er en særdeles anvendelig facilitet, hvis man f.eks. ønsker at hente det n'te element ud af en tabel, forudsat naturligvis, at start adressen gemmes i side nul.

Et eksempel på hvordan højopløst grafik kan bruges på VIC

Efter artiklen i sidste nummer, om brugerdefinerede tegn, er vejen nu banet for det sidste skridt mod højopløsningsgrafik.
Højopløsningsgrafik anvender nøjagtigt samme princip som brugerdefinerede tegn. Det man egentlig gør, er at fylde skærmen med hver af de 255 forskellige karakterkoder. (Kun halvdelen af skærmen kan bruges, med 8x8 karakterer). Karaktergeneratoren kan så bruges som et højopløst memory mappet område. (Memory mappet er et computerudtryk der betyder, at hver af det omtalte hukommelsesområdes bit, nøje svarer til de enkelte prikker i tegnene på skærmen). Hvis alle bytene i RAM karaktergeneratoren sættes til nul, så vil skærmen være blank. Hvis ét bit i en karakter sættes, vil det vise sig som en enkelt højopløst prik på skærmen. Forholdet mellem en enkelt prik på skærmen, placeringen i RAM karaktergeneratoren, og kodeværdien i hver af skærm-RAMmens pladser, vises i FIG. 1. Heraf ses også, at grundlaget for højopløst grafik, simpelthen består i at fylde skærmen med på hinanden følgende alfanumeriske værdier, (hele karaktersættet). resten er kun et spørgsmål om at foretage nogle beregninger for at sikre, at de rigtige bit sættes og slettes i de enkelte karakterkoder, der svarer til placeringen i skærm-RAMmen. I FIG. 2 ses et lille program, der er i stand til at foretage disse beregninger. Det består af to dele. I starten klargøres VIC chippens registre til en brugerdefineret karaktergenerator, sænker hukommelsens top for at beskytte karaktergeneratoren, placerer de korrekte tegn i skærm-RAMmen, og farve RAMmen, og sletter så indholdet af karaktergeneratoren. Den anden del, plotte delen, bruges hver gang, man ønsker at sætte eller slette en prik. Det er en rutine, der udfra givne X og Y koordinater, beregner hvilket bit i hvilket tegn, i den brugerdefinerede karaktergenerator, der skal sættes eller slettes. Bemærk her, at det område, der kan benyttes til højopløst grafik, kan variere fra ganske få tegnpladser til hele skærmen. For at kunne lave højopløst grafik over hele skærmen, må man indstille VIC chippen til 8x16 tegn, i stedet for de normale 8x8. Dette kræver, at RAM karaktergeneratoren udvides til 4K.

Skrives programmet, som det står, kræver det udvidelse med 3K RAM, men undlades REM-sætningerne og presses flere programlinier sammen, kan programmet bringes til at fungere i en VIC uden ekstra RAM.





ASCII kode-værdi gange 8 plus startadresse på karaktergenerator

```
10 REM ***************
20 REM *HØJ OPLØSNINGS
30 REM *GRAFIK EKSEMPEL
40 REM ***************
50 REM
60 REM
70 REM ***************
80 REM * BESKYT RAM
90 REM ***************
100 POKE51,255:POKE52,19
110 POKE55,255:POKE56,19:CLR
120 REM
130 REM ***************
140 REM *INIT VIC CHIP
150 REM ***************
160 PRINT"▯
170 POKE36867,128
180 POKE36865,60
190 F(0)=128:F(1)=64:F(2)=32:F(3)=16
200 F(4)=8:F(5)=4:F(6)=2:F(7)=1:F(8)=0
210 REM
220 REM **************
230 REM *SKRIV TEGN
240 REM *TAALMODIGHED!
250 REM **************
260 FORQ=0TO255
270 POKE7680+Q,Q: REM TEGN
280 POKE38400+Q,2: REM FARVE
290 NEXT Q
300 FORQ=5120TO5120+255*8
310 POKEQ,0: REM CLEAR GRAFIKOMRAADE
320 NEXTQ
330 REM
340 REM **************
350 REM * AKTIVERING
360 REM **************
370 POKE36869,253
380 POKE36866,PEEK(36866)OR128
390 POKE36867,150
400 REM
410 REM **************
420 REM *HOVED PROGRAM
430 REM **************
440 FORC=0TO175STEP2
450 REM
460 REM ***************
470 REM * FUNKTION
480 REM ***************
490 L=45+40*SIN(C/10)
500 REM
510 REM **************
520 REM *PLOTTE RUTINE
530 REM **************
540 A=5120
550 LR=L/8
560 LA=INT(LR)
570 A=A+(LA*176)
580 LR=(LR-LA)*8
590 CR=C/8
600 CA=INT(CR)
610 A=A+(CA*8)
620 CR=INT((CR-CA)*8)
630 POKEA,PEEK(A)ORF(CR)
    REM PLOT PUNKT
640 REM
650 NEXTC
660 REM
670 GETA$:IFA$=""THEN670
680 REM
690 REM **************
700 REM *NORMAL IGEN
710 REM **************
720 POKE36869,240
730 POKE36866,150
740 POKE36867,174
750 POKE36865,38
760 PRINT"▯
```

Sådan flyttes pointerne for at gøre plads til programmer i maskinkode

Et maskinkodeprogram, der skal befinde sig i RAM, placeres helst i toppen af RAMmen. Dette område benyttes af BASIC til at gemme tekststrenge i, og for at forhindre disse i at slette maskinkoden, må pointerne til toppen flyttes nedad. Top pointerne sættes under VIC's opstart til den højeste RAM adresse. Denne højeste adresse afhænger i hvert enkelt tilfælde af, hvor meget ekstra RAM der er tilføjet. Ved at sænke denne pointers værdi, kan der reserveres en blok af RAMmen, der så er fuldt beskyttet mod sletning, til f.eks. maskinkode. Operativsystemet vil så betragte denne pointers nye værdi som værende den højest mulig anvendelige RAM adresse, hvorved en sletning af maskinkoden undgåes. Pointeren gemmes som mindst betydende byte i 51, og mest betydende byte i 52. Følgende eksempel flytter pointeren til lokation 4096:

```
POKE 51,0:POKE 52,16
POKE 55,0:POKE 56,16
CLR
```

(0 + 16*256 = 4086)

Lokationerne 55 og 56 er pointeren, der peger på strengene, og denne pointer skal følge top-pointeren. CLR nulstiller alle variable, og sørger for, at alle andre pointere er indstillet korrekt. Man skal ved reservering af RAM til maskinkode være opmærksom på, om pladsen i forvejen er optaget til skærm eller tegngenerator.

USR og SYS

Af de to kommandoer, der benyttes til at kalde maskinkode med, USR og SYS, er SYS absolut den mest avancerede. Med SYS angiver man blot den decimale startadresse, så hvis maskinkoden starter i lokation 5000, overfører SYS 5000 kommandoen direkte. Variable kan overføres mellem BASIC og maskinkode ved hjælp af PEEK og POKE. Disse læser eller skriver værdier fra eller til lokationer, der i forvejen er valgt til det, og er fælles for de to programdele. Overførsel af variable på denne måde, er langt lettere end brugen af floating point variablen i USR(X) kommandoen. Det muliggør også overførsel af mere end én variabel, hvilket USR ikke gør. Den eneste betingelse, der er på en SYS kommando er, at den sidste maskinkode instruktion er en RTS, retur fra subrutine, der automatisk overfører kontrollen til BASIC igen.

En loader

Den letteste måde at indlæse et maskinkode program i computeren på, er at lade det være den del af et BASIC program, en simpel loader, hvor de enkelte maskinkoder er tal i DATA sætninger. Loaderen POKER så de enkelte tal ind i de korrekte lokationer. En anden metode er at anvende Mashine Language Monitor modulet. Hermed kan man indtaste maskinkode direkte i hexadecimal notation. Man kan også save og loade maskinkoden alene hermed. For at lette indtastningen og undgå den langsommelige »håndassemblering«, er der også simple assembler og disassembler kommandoer i modulet. Monitoren kan gemme et maskinkodeprogram ved kun at gemme den blok, maskinkoden fylder, mange gange hurtigere end nogen BASIC loader.

Den eneste ulempe ved at anvende monitoren til at loade og save maskinkode med er, at der så kræves en to-delt load, for at indlæse BASIC programmet og den tilhørende maskinkode. BASIC programmet kunne dog også saves med monitoren, således at hele området med BASIC program og maskinkode saves på én gang. Loadningen kunne derved gøres på én gang, men til gengæld ville den tage ret lang tid, afhængig af mængden af RAM.

Problemer med VICMON og en printer, der glemmer mellemrummene

Jeg har et problem med Commodores maskinkode-monitor. Der står i vejledningen at når man trykker på »X« går monitoren tilbage til basic. Det vil jeg vove at påstå ikke er tilfældet. Man kan ikke gøre ret meget efter at have lavet et maskinsprogsprogram. Der sker de mærkeligste ting og hvis i kan hjælpe med at oplyse hvilke registre der bliver ødelagt så man kan rette det til og køre i basic igen uden at skulle slukke for maskinen.

Det skal jeg nu ikke da jeg har monteret en »reset« knap (som burde være monteret ved køb af computeren) der resetter maskinen uden at slette maskinsprogsprogrammer og, man kan redde basic-programmer, hvis man husker at se efter de første bytes og register 45 og 46 inden man »runner« programmet.

Når man har trykket på knappen poker man bare de tal ind igen og man kan liste programmet og rette evt. fejl, men det er ikke tilfældet med maskinkodemonitoren.

Derfor vil jeg gerne have en forklaring på, hvad man gør i de tilfælde. På forhånd tak og jeg håber at maskinkode artiklerne fortsætter med nogle gode og brugbare eksempler. Det i manualen til monitoren, kan jeg ikke få til at gøre som der står der skal.

Man skulle jo tro, at man kunne få rettelsesblade ved forhandlerne efterhånden som tingene bliver rettet. Jeg synes, at det er meningsløst, at man skal købe et hav af blade for at få rettet fejl.

Jeg håber, at det kommer i VI & VIC. Det er et udmærket blad, som godt kunne være dobbelt så stort.

Jeg har også et andet problem:

Når man henter en tekst på diskettestationen, og skriver den ud på skærmen, er der ikke noget i vejen. Men når man så åbner til printeren i stedet for, så forsvinder alle space. Hvordan kan det være og hvad gør man for at beholde sine space. Man laver dem jo ikke for at printeren skal stjæle dem vel???

Dette er skrevet med data linier, for så er der ikke noget i vejen. Det virker bare. Jeg synes det er mystisk. Kan i give svar på disse ting i bladet eller også kan i sende svaret direkte. Men der er jo nok andre, der gerne vil have svarene på disse ting.

Den eneste forskel på udskrivningerne jeg laver er disse: OPEN4,3 til skærmen og OPEN4,4 til printeren. Udskrivning sker med print #4,A$(A).

Med venlig hilsen

Ole Andersen
Henning Smithsvej 32
9000 Aalborg

Løsningen på Ole Andersens problem ligger i anvendelse af E-kommandoen i forbindelse med VICMON. Det er omtalt i brugervejledningen for VICMON i kapitel 2 (på side 8), og de to instruktioner, der kan føre maskinen frem og tilbage mellem maskinsprog og BASIC, kan f.eks. se således ud: E 1000 for at komme væk fra BASIC og E 0000 for at komme tilbage igen.

Med hensyn til vanskelighederne med at få ordmellemrummene med på papiret, når en tekst fra diskette skal printes ud, er det ikke muligt at se, der bliver gjort noget forkert, men redaktionen vil tage direkte kontakt med Ole Andersen for at få opklaret, hvad det er der går galt. Vi vil senere dele vore erfaringer med VI & VIC's læsere.

VIC programmer

Lad andre få mulighed for at købe de programmer, du udvikler til VIC.

Rubrikannoncerne her i bladet koster 1 kr. pr. spaltemillimeter, og for overskuelighedens skyld bør annoncerne bygges op på samme måde, som det kan ses her på siden. Det vil sige, at annonceteksten skal indeholde oplysninger om nødvendigt udstyr, hvilket medium, programmet findes på (kassettebånd, disk eller cartridge), prisen - og naturligvis oplysninger om, hvor man kan købe programmet.

Brug kuponen til din annonce.

VIC program sælges

Programmets navn: ______________________

Beskrivelse: ______________________

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

Hardware: ______________ **Medium:** ______________

Pris: ______________________

Købes hos: ______________________

____

Kuponen sendes til VI & VIC, Kornager29, 7100 Vejle.

2 BOKSE

1. BOKS indeholder: Yatzy med store terninger. Klunse - du spiller mod VIC. Jack-pot med sandsynlighed. Børs - du handler aktier (max. 5 spillere).
2.BOKS indeholder: DK-karaktersæt. Gloser, hvor VIC lærer dig valgte gloser. Grafplotter, hvor VIC tegner en valgt funktion (med DM!).
PS. Skynd dig, så får du et spil **Guldfeber** gratis oveni!
Hardware: VIC-20 (med 3K ring!).
Medium: Kassettebånd.
Pris: Kun 100,- kr. pr. boks (+ ekstra spil).
Købes hos: Frank Dahl, Thorsvej 72, 8680 Ry.
Telf. (06) 89 25 24.

VIC-20 TIPS

Programmet giver forslag til tipstegn på grundlag af opdaterede data (16 uger tilbage) fra samtlige kampe fra den engelske 1. og 2. division. Endvidere kan programmet udskrive sypigetips.
Hardware: VIC-20 (3583 bytes).
Medium: Kassettebånd.
Pris: 165 kr.
Købes hos: Ann Hansen, tlf. (01) 15 74 50 eller (01) 21 64 09.

RUM-MONSTRE

Rum-monstre er et spændende rumspil, som går ud på, at du skal skyde grådige monstre ned med dine 3 laserkanoner, før de kommer hen og fortærer dine celler. Programmet er med højopløsningsgrafik og gode lydeffekter.
Hardware: 3.5K VIC-20.
Medium: Kassette.
Pris: 75,- kr.
Købes hos: Thomas Ehrenreich, Korsvejen 4, 8660 Skanderborg.

VIC-20

Hjemmecomputeren der vokser med opgaverne



❶ VIC-20 computer
Det private computersystem starter med VIC-20 - hjemmecomputeren fra Commodore. Verdens første fuldt udbyggede farvecomputer til **kr. 2.995,-**

VIC-20 kan udbygges til et enkelt eller mere avanceret system efter ønske og i takt med brugerens behov. Til privat brug. Til undervisning. Til forretningsbrug.

VIC-20 har skrivemaskinetastatur. 8 funktionstaster. 255 farvekombinationer. 64 grafiske tegn. Musikgenerator. Intern hukommelse op til 32K RAM. Tilsluttes alle TV-apparater og monitorer.

❷ VIC-1530 kassettestation
Det første skridt mange tager, når de udbygger deres VIC-computersystem. Kassettestationen kan bruges til datalager eller til overførsel af programmer. **Pris: kr. 1.035,-.**

❸ VIC-1540 diskettestation
Diskettestationen giver mulighed for det fulde udbytte af VIC-computersystemet. Lagerkapaciteten udvides til 170K bytes. Op til 4 diskettestationer kan kobles til samtidigt. Velegnet til programudvikling. Hurtig dataoverførsel. **Pris: kr. 6.495,-.**

❹ VIC-1515 printer
Rigtig papirprinter. 80 karakterer og en hastighed på 30 tegn/sek. **Pris: kr. 4.945,-.**

❺ VIC-1020 moder-modul
Dette modul er bl.a. nødvendigt for at udnytte VIC fuldt ud. Gør det muligt samtidig at tilslutte f.eks. 3 - 8 og 16K RAM, superexpander, etc. **Pris: kr. 2.280,-.**

❻ Udvidelse af hukommelse
VIC-20's hukommelseskapacitet kan udvides med 3K til 27K RAM.

❼ VICMON
Gør det muligt at programmere i maskinsprog. **Pris: kr. 400,-.**

❽ VIC-1211A Superexpander
Giver højopløst grafik og ekstra kommandoer til plotning, farvelægning, etc. **Pris: kr. 595,-.**

❾ VIC-1212 Programmers aid pack
Et godt hjælpeværktøj ved programmering. Giver 14 ekstra kommandoer. **Pris: kr. 495,-.**

❿ VIC-1311 joy stick
Til spil og lignende. **Pris: kr. 95,-.**

⓫ Brugerhåndbog
Der findes let forståelig dansk brugerhåndbog til VIC, som også på enkel og klar måde fortæller om BASIC. Hveranden måned udkommer brugerbladet VI & VIC, som kan købes i løssalg eller abonnement med VIC-nyheder.

Find nærmeste autoriserede VIC-forhandler på listen og kom ind og prøv VIC hjemmecomputeren.

Alle priser er incl. moms.

commodore COMPUTER

05 - 64 11 55 eller 01 - 88 15 05

ØERNE: Ballerup: Mic Kasseapparater, 02-655111. **Bornholm:** Krogh Hansens Radio & TV, 03-998127. **Herlev:** W. Rolf Pedersen, 02-915511. **Hillerød:** Bo-el Data, 02-253131. **Holbæk:** Papirgården, 03-433003. **København:** BN Elektronic, 01-811900 og 01-184555. CR-Commander Radio, 01-343422. L. A. Elektronik Kbh., 01-551540. Betafon Radio, 01-310273. Samfundslitteratur, 01-351942. Poly Data Microcenter, 01-420705. W. Rolf Pedersen, 01-137056. **Lyngby:** BN Elektronik, 02-881900. Polyteknisk Boghandel, 02-881488. **Maribo:** JD Totalinformation, 03-881700. **Odense:** Dansk Totalinformation, 09-126488. EDB Butikken, 09-147660. Fl. Kjerulff, 09-135480. **Otterup:** Werners Radio Elektronik, 09-823333. **Ringsted:** Ahrent Flensborg, 03-610011. **Rødovre:** Datacare, 01-705858. **Slagelse:** Vest Sjællands Mikrodata, 03-534707. **St. Heddinge:** JE Kontorcenter, 03-703332. **Svendborg:** Jens Bach Kontordata, 09-212788. **Viby Sj.:** Baagøe Radio, 02-394100. **Vordingborg:** Kontorforsyningen, 03-770170. **GRØNLAND: Godthåb:** Godthåb Elektronikservice, 21865. **Julianehåb:** Musikhuset Julianehåb, 01-438133. **JYLLAND: Aalborg:** Knud Engsig Kontorudstyr, 08-126666. H. C. Elektronic, 08-142314. **Esbjerg:** Kurt Øland, 05-125299. **Fredericia:** Semicap Data, 05-931565. **Gram:** Teknik Huset, 04-822711. **Grenå:** Djursland Data, 06-326465. **Haderslev:** H. C. Reimers, 04-523714. **Hadsten:** Hadsten Elektronik, 06-980408. **Herning:** Logic Design, 07-221300. **Hjørring:** Norad, 08-960188. **Holstebro:** Pe-Co Computer System, 07-414646. **Horsens:** Dansk Computer Center, 05-611110. **Kolding:** Hauge Rasmussen Foto, 05-520070. **Randers:** Djursland Data, 06-434200. **Ringkøbing:** LP Musik, 07-322511. **Skanderborg:** Bildata, 06-525344. **Skive:** Elektroniklageret, 07-526177. **Struer:** Helmholt Elektronik, 07-852611. **Sønderborg:** G. P. Kontorsystemer, 04-424546. **Vejle:** 2R Data, 05-831600. **Århus:** Akademisk Boghandel, 06-128844. Clemens Papirteknik, 06-133922. Editor, 06-127720.

Hedelund Marketing, Vejle